Building Master Tables for Claims, Commission, and Premium in an Insurance Database

Hi! That's a great project—centralizing data into master tables (e.g., for reporting, analytics, or ETL processes) can improve efficiency in an insurance company. Master tables typically aggregate or consolidate data from original/source tables (e.g., transactional logs, policy databases, or claim systems). To link them effectively, you'll need **common keys** (shared identifiers across tables) and **foreign keys** (references from one table to another's primary key) to maintain referential integrity and enable joins.

Based on standard insurance database schemas (e.g., involving policies, clients, agents, and transactions), I'll suggest possible keys for each master table. These are derived from common insurance data models, assuming your original tables include entities like policies, customers, claims, payments, and agents. Adjust based on your specific schema—e.g., if you have a "Policy" table as a central hub, many keys will reference it.

General Assumptions and Tips

- **Primary Keys**: Each master table should have a unique ID (e.g., auto-incremented).
- **Common Linking Strategy**: Use **Policy_ID** as a central hub, since claims, commissions, and premiums all tie back to policies. Other universals: Client_ID, Agent_ID, Date fields.
- **Database Best Practices**: Use SQL (e.g., PostgreSQL/MySQL) for constraints; enforce foreign keys to prevent orphans. For large data, consider indexing keys for faster joins.
- **If Data is Missing**: Add surrogate keys or ETL processes to generate them during master table build.

Now, for each master table:

1. Claims Master Table

This table aggregates claim-related data (e.g., claim status, amounts, dates). Original tables might include: Claim_Details, Policy_Claims, Payouts.

Possible Common/Foreign Keys to Link Original Tables:

- **Policy_ID** (Foreign Key): Links to Policy table (original source for policy details).

Ensures claims tie to specific insurance policies.

- **Client_ID** or **Policyholder_ID** (Foreign Key): References Customer/Client table (for claimant info).
- **Claim_ID** (Primary Key in master; Foreign Key from sub-tables like Claim_Events or Adjuster_Logs).
- **Agent_ID** or **Broker_ID** (Foreign Key): From Agent table (if agent handled the claim).
- **Date_Key** (e.g., Claim_Date, Approval_Date): Links to Date dimension table for time-based analysis.
- **Product_ID** or **Coverage_Type_ID** (Foreign Key): From Product table (e.g., auto vs. health insurance).
- **Payout_ID** (Foreign Key): Links to Payments table (for claim settlements).

How to Build/Link:

- ETL Query Example (SQL): `SELECT c.Claim_ID, p.Policy_ID, cl.Client_ID, ... FROM Claims c JOIN Policies p ON c.Policy_ID = p.Policy_ID JOIN Clients cl ON p.Client_ID = cl.Client_ID;`
- Benefit: Enables queries like "Claims by Policy Type."

2. Commission Master Table

This consolidates agent/broker commissions (e.g., earned amounts, dates). Original tables: Commission_Payments, Sales_Logs, Agent_Contracts.

Possible Common/Foreign Keys to Link Original Tables:

- **Policy_ID** (Foreign Key): Ties commissions to sold policies (from Policy table).

- **Agent_ID** or **Broker_ID** (Foreign Key): Primary link to Agent table (who earned the commission).
- **Client_ID** (Foreign Key): From Customer table (for the policy buyer).
- **Commission_ID** (Primary Key in master; Foreign Key from Payment_Details).
- **Date_Key** (e.g., Commission_Earned_Date, Payment_Date): Links to Date table.
- **Product_ID** (Foreign Key): From Product table (commission rates vary by product).
- **Transaction_ID** (Foreign Key): From Sales or Premium_Payments table (for commission triggers).

How to Build/Link:

- ETL Query Example: `SELECT com.Commission_ID, p.Policy_ID, a.Agent_ID, ... FROM Commissions com JOIN Policies p ON com.Policy_ID = p.Policy_ID JOIN Agents a ON com.Agent_ID = a.Agent_ID;`
- Benefit: Tracks "Commissions by Agent/Product" for performance audits.

3. Premium Master Table

This aggregates premium payments (e.g., amounts, due dates, status). Original tables: Premium_Payments, Billing_Logs, Policy_Premiums.

Possible Common/Foreign Keys to Link Original Tables:

- **Policy_ID** (Foreign Key): Core link to Policy table (premiums are policy-specific).

- **Client_ID** (Foreign Key): From Customer table (payer info).
- **Premium_ID** or **Payment_ID** (Primary Key in master; Foreign Key from Transaction_Logs).
- **Agent_ID** (Foreign Key): From Agent table (if agent collected/sold).
- **Date_Key** (e.g., Due_Date, Payment_Date): Links to Date table.
- **Product_ID** (Foreign Key): From Product table (premium rates by product).
- **Installment_ID** (Foreign Key): For multi-payment premiums, links to Payment_Schedule.

How to Build/Link:

- ETL Query Example: `SELECT prem.Premium_ID, p.Policy_ID, cl.Client_ID, ... FROM Premiums prem JOIN Policies p ON prem.Policy_ID = p.Policy_ID JOIN Clients cl ON p.Client_ID = cl.Client_ID;`
- Benefit: Enables "Premiums by Product/Client" for revenue forecasting.

Linking All Three Master Tables (Cross-Theme Integration)

To connect Claims, Commission, and Premium masters (e.g., for holistic analysis like "Claims Impact on Premiums"):

- **Shared Keys**: Use **Policy_ID** as the universal join (all tie to policies). Add **Client_ID** and **Product_ID** for multi-table queries.
- **Example SQL Join**: `SELECT * FROM Claims_Master c JOIN Premium_Master pr ON c.Policy_ID = pr.Policy_ID JOIN Commission_Master com ON pr.Policy_ID = com.Policy_ID;`
- **Database Design Tip**: Create a **Data Warehouse** with fact tables (masters) and dimension tables (Policy, Client) for efficient linking via star schema.

Implementation Tips

- **Tools**: Use SQL Server/PostgreSQL for database; Python (pandas/SQLAlchemy) or ETL

tools (Talend/Alteryx) for building.

- **Best Practices**: Enforce foreign keys with constraints; index keys for speed; handle nulls/duplicates during ETL.
- **Challenges**: Data quality (e.g., mismatched IDs)—use validation scripts.

If you share more details (e.g., original table schemas), I can refine the keys or provide SQL code!